



[THE BLAZINGCORE SERIES]

WITH SUPERIOR NUMBER CRUNCHING ABILITIES AND PERIPHERAL HANDLING ON OUR CUSTOM EMBEDDED OS,
RAPID PROTOTYPING IS NOW EASY... AND BLAZING FAST.

INTRODUCTION

The BlazingCore(BC) Family Series is a 32-bit hardware and software series, comprising of an entire range of boards built around the PIC32MX chip by Microchip; all running on the BCore Operating System. The BCore IDE provides the programming platform for writing programs in .NET syntax language (VB.Net & C#.NET) with built-in BCore OS commands to control devices and supports most standard protocols eg.(UART,SPI,I2C,PWM,ADC...), as well as built-in libraries for Servo Motors, Graphic LCDs, and OLED Displays.

TECHNICAL OVERVIEW

BLAZINGCORE (BC)

Core	BCore100	
Chip	PIC32MX695F512L	
Operating System	BlazingCore	
Operating Voltage	3.3V	
I/O Sink/Source Current	18mA	
Range	Max Rating: 25mA	
Pins	32bit	
Package	100	
PIC Family	TQFP	
Speed	PIC32MX	
Physical I/O Pins	5 Million Lines of Code per second	
User Memory	67	
External Memory	Data Flash	SD
Timer/s	5	
UART	2	
PWM Pins	5	
ADC Pins	16	
ADC Resolution	10bit	
SPI	2	

FEATURES

High performance 32-bit RISC CPU

- Based on Microchip's PIC32MX Chip
- MIPS32® Architecture
- BlazingCore™ OS Onboard
- External BCore Memory Support for Data Flash, MMC (SD – Compatible)
- 5 Million Lines of Instructions per Second
- 80MHz Clock Speed

Microcontroller features (BCore100)

- 3.3V Low Power Consumption
- 110K Onboard SRAM Memory
- Up to 16MB External Data Flash Memory

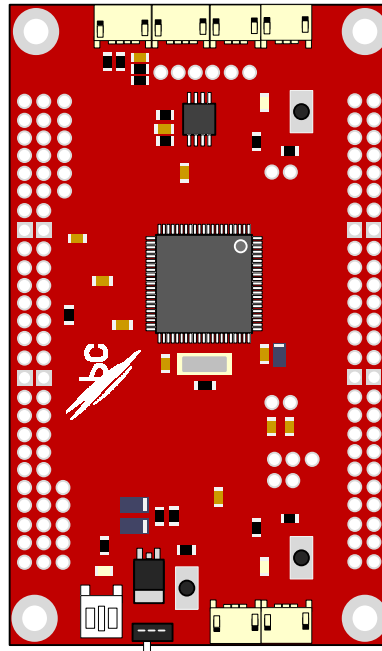
Note: External Memory Card Support for MMC (SD Compatible) (up to 16Gb) on OLED interfacing board allows user to add big files.

Peripheral Features

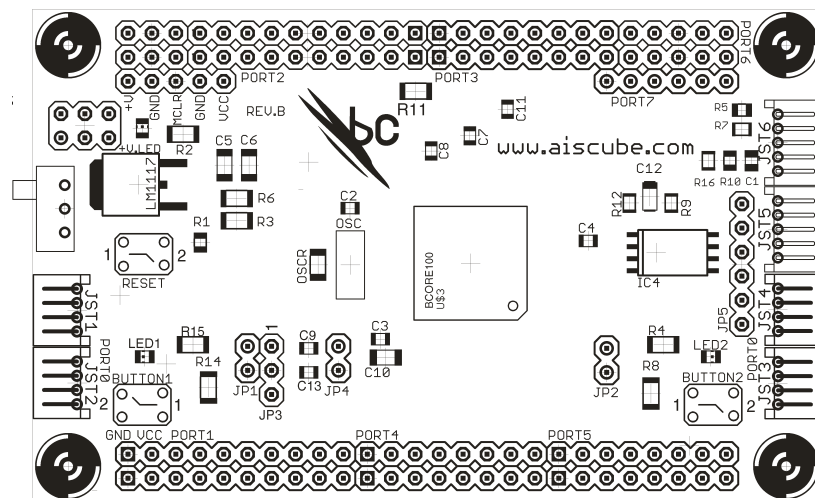
- 67 Physical I/O Pins
- High Current Sink/Source 18mA/18mA on all I/O pins
- Two UART Modules
- Five PWM Outputs
- High Speed I/O Pins Capable of Toggling at up to 10MHz
- 5V Tolerant Input Pins (Digital Only)

Analog Features

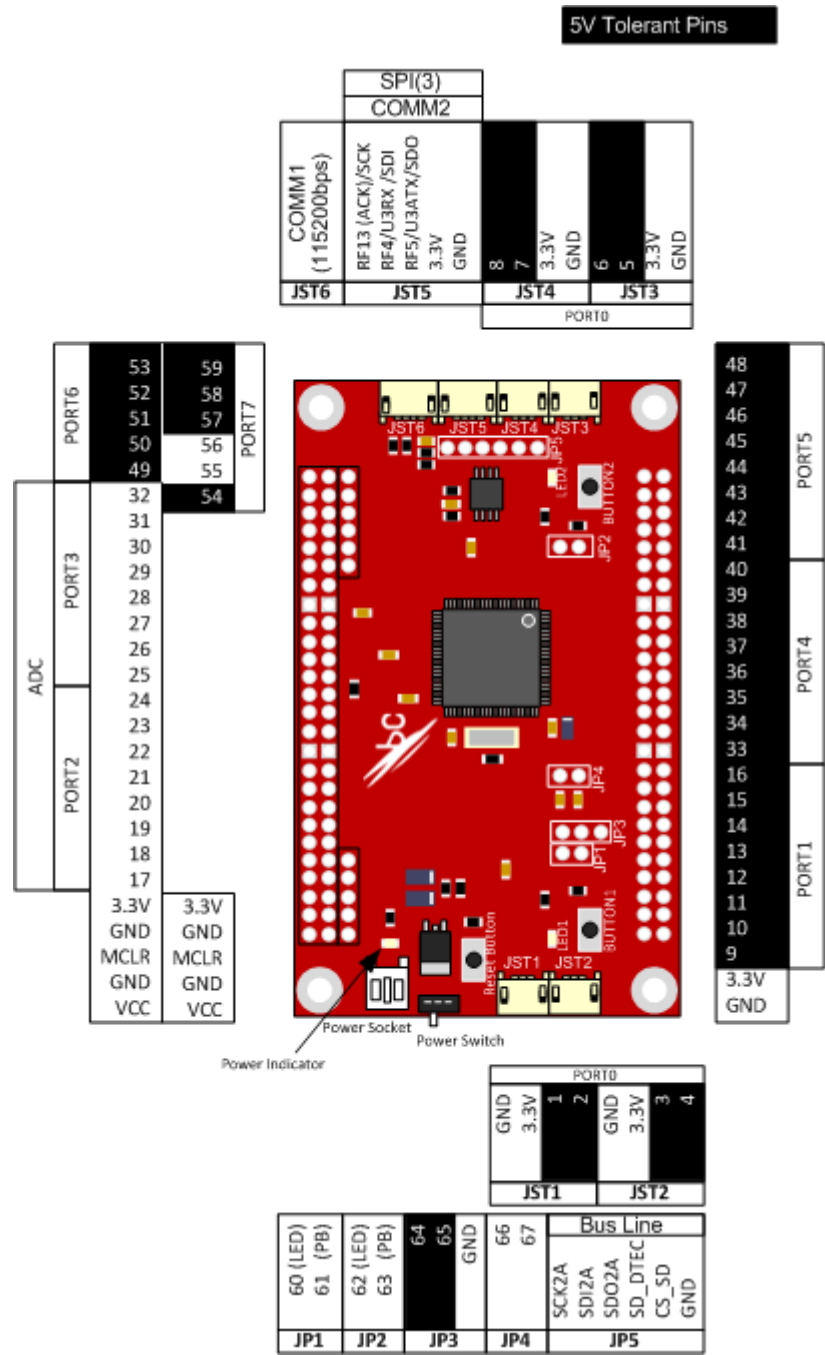
- 16 Channel 10-bit Resolution Analog-to-Digital Converter
 - 1Msps conversion rate



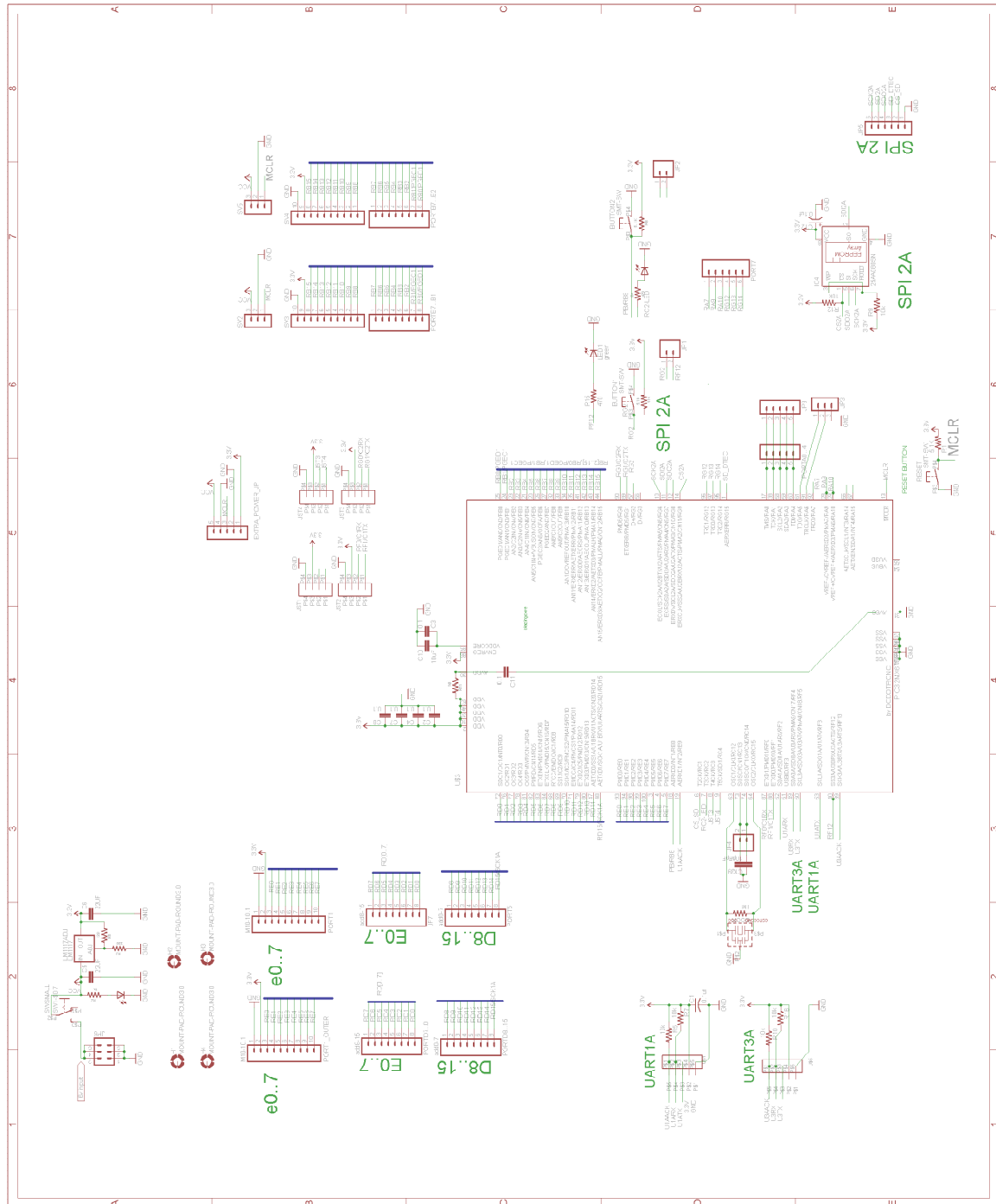
BCore100



PINOUT



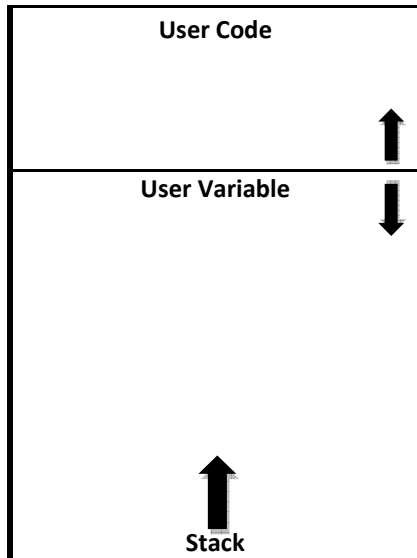
SCHEMATIC



MEMORY MANAGEMENT

ONBOARD MEMORY

SRAM – 110K



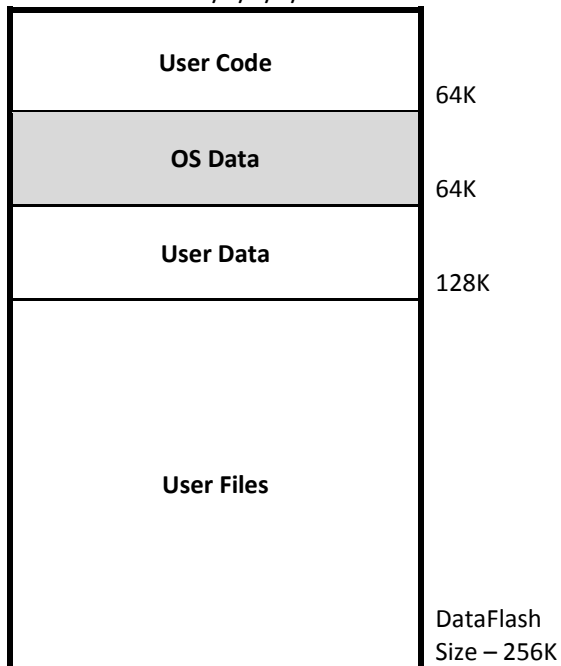
There is a total of 110K of SRAM Memory for User Code, User Variable and the Program Stack. The amount of memory allocated for the code and variable is user defined.

Users are able to define the amount of memory to allocate for code and data within the IDE using the configurations page.

The Program Stack grows and shrinks during program run time as variables are declared, assigned and disposed.

EXTERNAL MEMORY – BCore100

DataFlash Size – 1/2/4/8/16MB



The BCore100 uses a DataFlash chip for its external memory. BCore Programs are stored here.

DataFlash Memory size supported are 1/2/4/8/16MB.

BCore allocates fixed regions within the DataFlash as illustrated on the left;

- 64K each for User Code and OS Data;
- 128K for User Data;
- And the remaining memory (DataFlash Size – 256K) for User Files(.bmp/.wav/.txt Files)

Note: By default, BCore100 Boards ship with 1MB of external dataflash chip.

I/O PINOUT DESCRIPTION

Pins	PIC32MX695F512L	BCore100 (REV.B)				
1	RA14	GIO	JST1	PORT0	PORTA	
2	RA15	GIO				
3	RF0	GIO	JST2		-	
4	RF1	GIO				
5	RG0	GIO	JST3			
6	RG1	GIO				
7	RC3	GIO	JST4			
8	RC4	GIO				
9	RE0	GIO		PORT1		PORTE
10	RE1	GIO				
11	RE2	GIO				
12	RE3	GIO				
13	RE4	GIO				
14	RE5	GIO				
15	RE6	GIO				
16	RE7	GIO				
17	RB15	GIO	ADC	PORT2	PORTB	
18	RB14	GIO	ADC			
19	RB13	GIO	ADC			
20	RB12	GIO	ADC			
21	RB11	GIO	ADC			
22	RB10	GIO	ADC			
23	RB9	GIO	ADC			
24	RB8	GIO	ADC			
25	RB7	GIO	ADC	PORT3		
26	RB6	GIO	ADC			
27	RB5	GIO	ADC			
28	RB4	GIO	ADC			
29	RB3	GIO	ADC			
30	RB2	GIO	ADC			
31	RB1	GIO	ADC			
32	RB0	GIO	ADC			
33	RD7	GIO		PORT4	PORTD	
34	RD6	GIO				
35	RD5	GIO				
36	RD4	GIO				
37	RD3	GIO				
38	RD2	GIO				
39	RD1	GIO				
40	RD0	GIO	(SPI1) SDO1			
41	RD8	GIO		PORT5		
42	RD9	GIO	(SPI1) SS1			
43	RD10	GIO	(SPI1) SCK1			
44	RD11	GIO				
45	RD12	GIO				
46	RD13	GIO				
47	RD14	GIO	(SPIx) SS1A			
48	RD15	GIO	(SPIx)SCK1A			

I/O PINOUT DESCRIPTION

49	RA0	GIO	PORT6	PORTA
50	RA1	GIO		
51	RA2	GIO		
52	RA3	GIO		
53	RA4	GIO		
54	RA7	GIO	PORT7	-
55	RA9	GIO		
56	RA10	GIO		
57	RG12	GIO		
58	RG13	GIO		
59	RG14	GIO		
60	RF12	LED1		
61	RG2	BUTTON1		
62	RA5	GIO		PORTA
63	RA6	GIO		
64	RC13	GIO		-
65	RC14	GIO		
66	RC2	LED2		
67	RE8	BUTTON2		

ADDRESSABLE PORT REGISTERS

The following ports are addressable with pins corresponding to the registers of the PIC32 Chip.

	LSB															MSB	BITS PIN NO.
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
PORTA	49	50	51	52	53												
PORTB	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	
PORTD	40	39	38	37	36	35	34	33	41	42	43	44	45	46	47	48	
PORTE	9	10	11	12	13	14	15	16									

DEVICE COMMUNICATION

OVERVIEW

	BCore100
UART	2
SPI	2

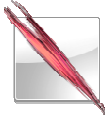
DETAILED INFORMATION

		PIC32MX695F512L
UART1	ACK	RE9
	RX	U1RX/RF2
	TX	U1TX/RF8
UART2	ACK	RF13
	RX	U2RX/SDA2/RF4
	TX	U2TX/SCL2/RF5
SPI1	SCK	SCK1/RF6
	SDI	U1RX/SDI1/RF2
	SDO	U1TX/SDO1/RF3
SPI2	SCK	SCK2/RG6
	SDI	SDI2/RG7
	SDO	SDO2/RG8

*Note: ¹ Items shaded indicates that these pins are reserved for specific hardware purposes.

²As indicated above, most of the communication pins are shared. Users should take note that only one form of communication can be used at any one time.

BCORE IDE



The BCoreIDE is a **structured programming environment**, with support for global and local structures, subroutines, variables and APIs for advance users, as well as file streams exposed for users to carry out data manipulation. The BCore OS also boasts of built-in **libraries (OS Commands)** that **natively** handle control over I/O peripherals, external displays (e.g. OLED Displays/Graphic LCDs), communication protocols, as well as external memory.

.NET SYNTAX LANGUAGE

The BCoreIDE provides the programming platform for the BCore Boards using Microsoft®'s **.NET Syntax Language**. At this point of writing, it currently supports the **VB.NET language**, although **C#.NET language** support is underway.

Users should take note that while the syntax accepted is meant to be similar to the .NET language, the platform is essentially an embedded system and not a full-fledged .NET platform running on the Windows® OS. As such, the language is streamlined to better suit the embedded platform.

BCORE OS COMMANDS

The BCore OS Commands are native libraries that address the embedded side of the BCore Chip. They are separate from the .NET Syntax Language and are usually denoted by its syntax colour (dark blue), as opposed to the .NET syntax colour (blue). The OS Commands are used when dealing with the IO peripherals, system components, device communication protocols, and for performing bit, byte and word manipulations. When custom control is required and not supported natively by the BCore OS nor amongst the many libraries available for common devices shipped with the IDE, you will find that modifying codes from the library similar to your device or software bit-banging is easily achieved.

COMMENT/REMARK FORMAT

An apostrophe (') is used to denote a comment.

Code:

```
01. 'This is a remark
02. Dim S As String
03. S = " 'This is not a remark"
```

All characters to the right of the apostrophe are usually ignored by the compiler, unless the apostrophe is embedded inside a string literal.

VARIABLE DECLARATION

`Dim <Identifier> As <Data_Type>`

All variables must be declared before they're used. Implicit declarations are not allowed.

For variables declared in module-level code:

`[Public | Private | Dim] <variable> As <Data_Type>`

Public variables are global and visible throughout the entire program. Private variables are visible only in the module in which they appear. The default is private – that is, if Dim is used for a module level variable, the variable is private.

For variables declared inside a subprogram:

`Dim <variable> As <Data_Type>`

Variables declared inside a subprogram are visible only inside the subprogram.

Code:

```
01. Public Distance As Integer ' Module-level variable, global
02. Private Temperature As Single ' Module-level variable local to module
03. Sub Main()
04.     Dim PinNumber As Byte 'Variable is local to this subprogram
05. End Sub
06. Sub ReadPin()
07.     Dim PinNumber As Byte 'Variable is local to this subprogram
08. End Sub
```

VARIABLE IDENTIFIERS

Identifiers must start with a letter, and all other characters must be letters, digits or underscores.

An identifier can be up to 255 characters long, and all characters are significant.

Identifiers are not case sensitive. (E.g. Identifiers xyz, XYZ and xYz are equivalent.)

Reserved words like keywords are not allowed to be used as an identifier.

DATA TYPES

BCore currently supports these data types.

Data Type	Memory Size	Content
Boolean	8bits	True/False
Byte	8bits / 1 Byte	0 – 255
Integer	16bits / 2 Bytes	-32 768 to 32 767
Long	32bits / 4 Bytes	-65 536 to 65 535
Single		
String	64Bytes	Max: 64 Chars
Point/Points()	32bit; (16bit X, 16bit Y)	X & Y (Point.X , Point.Y)
Rectangle/Rectangles()		2 Points (Point1, Point2)

CONSTANTS

For constants declared in module-level code:

```
[Public | Private] Const constant_name As <Data_Type> = <literal>
```

By default, a declaration is private.

LABELS

Labels serve as targets for the 'goto' statements. Mark the desired statement with a label and colon like this:

```
label_identifier : statement
```

Code:

```
01. Dim Count As Integer
02. Public Sub Main()
03.     Count = 0
04.     AGAIN:
05.         Count = Count + 1
06.         If Count <> 5 Then
07.             Goto AGAIN
08.         End If
09.     End Sub
```

EXPRESSIONS

An expression is a sequence of operators, operands, and punctuators that returns a value.

Code:

```
01. ABS(variable)
02. SETBIT(variable, position)
03. CLRBIT(variable, position)
04. GETBIT(variable, position)
05.
06. HIGH(variable)
07. LOW(variable)
08.
09.
```

ARRAYS

An array represents an indexed collection of elements of the same type (called the base type).

Note: Maximum dimension is 3.

Supports;

- Global and local declaration
- Passing by reference to routines/subprograms

Code:

```
01. Dim ArrayName(array_length) As Integer '1 dimension
02. Dim ArrayName(array_length_1, array_length_2) As Integer '2 dimensions
03. Dim ArrayName(array_length_1, array_length_2, array_length_3) As Integer '3 dimensions
```

NOTE: While the maximum dimension for declaring an array is 3, please take note that the maximum length allowed for the 2nd and 3rd dimension *e.g.* (*array_length_2, array_length_3*) is 255. There is no limit for the 1st dimension of the array, but onboard SRAM memory pertaining to the respective chip you are using should be taken to mind. The array length should not exceed the SRAM Memory available.

STRINGS

A string represents a sequence of characters and is known to be a specialised array of characters. Strings are always declared to be 64 characters long, which is the maximum number of characters it can hold.

```
[Public | Private | Dim] <variable> As String
```

STRING METHODS

The following table lists the methods that are available with the use of a string, and its description.

Method	Description
[+]Add	Add: Plus operator enables strings to be concatenated.
Compare	Compares two strings. Returns True if strings match, False if it does not.
Chr	Converts
Cstr	Convert non-string data type variables to string
CstrHex	Convert non-string data type variables to its hexadecimal equivalent in a string.
Len	Returns length of String
Mid	Returns a string containing a specified range of characters from a string.

[+] ADD

Strings can be concatenated by adding them together using the plus operator. Additions of strings are also allowed between strings of different memory sources, as well as in conjunction with other string methods.

Code:

```
01. Dim StringA, StringB, StringC As String
02. Public Sub Main()
03.     StringA = "This Is "
04.     StringB = "A String!"
05.     StringC = StringA + StringB
06.     Debug.Print StringC
07. End Sub
```

Output :

```
01. This Is A String!
```

.COMPARE

Compares two strings.
Returns *True* if strings match, *False* if it does not.

```
Variable = String.Compare(string_A, string_B)
```

CHR

Returns the character associated with the specified character code as string.
Valid range for *character code* is 0 through 255.

```
Chr (Character Code)
```

Code:

```
01. Dim STR1 As String
02. STR1 = "TEST " + CHR(66) + CHR(67)
03. DEBUG.PRINT STR1
```

Output :

```
01. TEST BC
```

CSTR

Convert non-string data type variables to string.

`Cstr(variable)`

Code:

```
01. Dim I As Integer
02. Public Sub Main()
03.     I = 100
04.     Debug.Print Cstr(I)
05. End Sub
```

Output :

```
01. 100
```

CSTRHEX

Convert non-string data type variables to its hexadecimal equivalent in a string.

`Cstr(variable)`

Code:

```
01. Dim K As Long
02. Public Sub Main()
03.     K = IO.Portb()
04.     Debug.Print CstrHex(K)
05. End Sub
```

.LEN

Returns the number of characters within a specified string.

`String.Len(String)`

Code:

```
01. Public Sub Main()
02.     Dim str As String
03.     Str = "0123456789"
04.     Dim length As Integer
05.     Length = String.Len(str)
06.     Debug.Print "Length of Str = ";cstr(length)
End Sub
```

Output :

```
01. Length of Str = 10
```

MID

Returns a string containing a specified range of characters from a string.

Note: Unspecified length will return characters from Index to end of string.

`MID(String, Start Index, Length)`

Code:

```
01. Public Sub Main()  
02.     Dim MIDtest As String  
03.     Dim a As String  
04.     Dim b As String  
05.     Dim c As String  
06.  
07.     MIDtest = "First Middle Last"  
08.     a = Mid(MIDtest, 1, 5)  
09.     b = Mid(MIDtest, 7, 6)  
10.     c = Mid(MIDtest, 7)  
11.  
12.     Debug.Print "a = ";a  
13.     Debug.Print "b = ";b  
14.     Debug.Print "c = ";c  
15.  
16. End Sub
```

Output :

```
01. a = First  
02. b = Middle  
03. c = Middle Last
```

STATEMENTS

Operators describe and perform an operation between two or more values.

Statement format

A statement begins at the beginning of a line of text and terminates at the end of a line of text.

IF Statement

Conditionally executes a group of statements, depending on the value of an expression.

```
If <condition> Then
    [ statements ]
[ ElseIf <condition> Then
    [ statements ] ]
[ Else
    [ statements ] ]
End If
-or-
If condition Then [ statements ]
```

Select Case Statement

Runs one of several groups of statements, depending on the value of an expression.

```
Select Case <test expression>
Case <expression>
    [ statements ]
[ Case <expression>
    [ statements ] ]
[ Case Else
    [ statements ] ]
End Select
```

ITERATION STATEMENTS (LOOPS)**For Statement**

Repeats a group of statements a specified number of times.

```
For counter = start To end [ Step value ]
    [ statements ]
    [ Exit For ]
    [ statements ]
Next
```

Code:

```
01. Dim i As Integer
02.
03. For i = 0 To 3
04.     Debug.Print CSTR(i)
05. Next
```

Output :

```
01. 0
02. 1
03. 2
04. 3
```

Do Statement

Repeats a block of statements while a Boolean condition is True or until the condition becomes True.

```
Do { While } <condition>
    [ statements ]
    [ Exit Do ]
    [ statements ]
Loop
-or-
Do
    [ statements ]
    [ Exit Do ]
    [ statements ]
Loop { Until } <condition>
```

Code:

```
01. Dim i As Integer
02. i = 0
03.
04. Do
05.     Debug.Print Cstr(i)
06.     i = i + 1
07. Loop Until (i = 3)
```

Output :

```
01. 0
02. 1
03. 2
04. 3
```

OPERATORS

Operators describe and perform an operation between two or more values.

ARITHMETIC OPERATORS

Arithmetic operators are used to perform mathematical computations. They have numerical operands and return numerical results.

The arithmetic operators are addition (+), subtraction (-), multiplication (*), division (/), integer division (\), modulus (Mod), negation (-) and exponentiation (^).

Order of precedence for arithmetic operators follow the rules of precedence from basic math, which is, left to right.

Operator	Operation
^	Exponentiation
-	Negation
*, /	Multiplication and division
\	Integer Division
Mod	Modulus
+, -	Addition and subtraction

Table above lists operators and operations from highest precedence to the lowest.

RELATIONAL OPERATORS

Use relational operators to test equality or inequality of expressions. All relational operators return TRUE or FALSE.

Operator	Operation
=	equal
<>	not equal
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal

All relational operators associate from left to right.

BITWISE OPERATORS

Use the bitwise operators to modify the individual bits of numerical operands.

Operator	Operation
and	bitwise AND; compares pairs of bits and generates a 1 result if both bits are 1, otherwise it returns 0
or	bitwise (inclusive) OR; compares pairs of bits and generates a 1 result if either or both bits are 1, otherwise it returns 0
xor	bitwise exclusive OR (XOR); compares pairs of bits and generates a 1 result if the bits are complementary, otherwise it returns 0
not	bitwise complement (unary); inverts each bit
<<	bitwise shift left; moves the bits to the left, it discards the far left bit and assigns 0 to the right most bit.
>>	bitwise shift right; moves the bits to the right, discards the far right bit and if unsigned assigns 0 to the left most bit, otherwise sign extends

BOOLEAN OPERATORS

Operator Operation

AND	logical AND
OR	logical OR
XOR	logical exclusive OR (XOR)
NOT	logical negation

Boolean operators associate from left to right. The negation operator 'not' associates from right to left.

OVERALL OPERATOR PRECEDENCE

(Highest)	[1] Not
	[2] * \ Mod And
	[3] + - Or Xor &
(Lowest)	[4] = > < <> <= >=

SUB MAIN()

All projects must have a procedure “Main()” in the project. This is the starting point of the project.

Note: Only **ONE** Sub Main() is allowed in a single project.

Code:

```
01. Public Sub Main()
02.
03. End Sub
```

PROCEDURES

Procedures and functions, referred to collectively as routines, are self-contained statement blocks that can be called from different locations in a program.

A **function** is a routine that **returns a value** when it is executed. A **procedure** is a routine that **does not return a value**.

Once these routines have been defined, you may call them once or multiple times. A procedure is called upon to perform a certain task, while a function is called to compute and return a certain value.

SUB PROCEDURES

```
[Private|Public] Sub procedure_name (arguments)
[statements]
End Sub
```

You may exit a procedure at any point in the routine by using an `Exit Sub` statement.

Code:

```
01. Private Sub GetAnswer(ByRef b As Boolean)
02.
03.     If (b = TRUE) Then
04.         Exit Sub
05.     Else
06.         'do something
07.     End If
08.
09. End Sub
```

FUNCTIONS

```
[Private|Public] Function function_name (arguments) As <Data_Type>
[statements]
[Return Variable/Value]
End Function
```

The function returns a value. This is achieved by using the ‘Return’ command followed by the value to return within the Function. You may also exit a function by using an `Exit Function` statement.

Code:

```
01. Public Function F(ByVal i As Integer) As Integer
02.     If (i = 3) Then
03.         Return 92
04.         Exit Function
05.     End If
06.     Return i + 1
07. End Function
```

PASSING PARAMETERS TO ROUTINES/SUBPROGRAMS

Parameters can be passed to a subprogram by reference ([ByRef](#)) or by value ([ByVal](#)).

Passing by reference -- if you pass a parameter by reference, any changes to the parameter will propagate back to the caller. Pass by reference is the default.

Passing by value -- if you pass a parameter by value, no changes are allowed to propagate back to the caller. With a few exceptions, if an argument is passed by value, a copy is made of the argument, and the called subprogram operates on the copy. Changes made to the copy have no effect on the caller.

One exception is for string parameters passed by value. For efficiency reasons, a copy of the string is not made. Instead, the string is write-protected in the called subprogram, which means you can neither assign to it nor pass it by reference to another subprogram. You are allowed to pass it by value to another subprogram, however.

The other exception is for types `Unsigned Integer` and `Unsigned Long`, which are treated similarly – these parameters are write-protected in called subprograms.

Actual vs. formal parameters -- the type and number of the actual parameters must match that of the "formal" parameters (the formal parameters appear in the subprogram declaration). If there is a type mismatch, the compiler will declare an error. It will not do implicit type conversions.

Restrictions on passing mechanisms:

- Scalar variables and array elements can be passed by value or by reference.
- Since arrays are global they cannot be passed into a function or subroutine.
- Numeric expressions and numeric literals can be passed by value but not by reference. The same applies to Boolean expressions and Boolean literals.

	ByRef (By Reference)	ByVal (By Value)
Boolean	✓	✓
Byte	✓	✓
Integer	✓	✓
Long	✓	✓
Single	✓	✓
String	✓	✓
Array	✓	
Structure	✓	
Point/Points()	✓	
Rectangle/Rectangles()	✓	

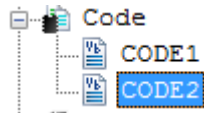
STRUCTURES

```
[Public | Private] Structure Structure_Name
    [Public | Dim] <variable> As <data_type>
End Structure
```

A structure declaration starts with the **Structure** statement and ends with the **End Structure** statement. The structure statement supplies the name of the structure, which is also the identifier of the data type the structure is defining. Other parts of the code can use this identifier to declare variables, parameters, and a function to return values of this structure's data type.

Structures are declared on its own, not contained within any procedures.

The declarations between the **Structure** and **End Structure** statements define the members of the structure.



Structures can be declared globally and accessed across different modules. The following code illustrates this.

A public structure is declared in Module “CODE2”.

Code:

```
01. 'CODE2
02. Public Structure structUser
03.     Public Name As String
04.     Public Gender As Boolean '0 = Male; 1 = Female
05.     Public Age As Integer
06. End Structure
07.
```

This structure is then declared and accessed through the main program in Module “CODE1”.

Code:

```
01. 'CODE1
02. Public USER As CODE2.structUser
03.
04. Public Sub Main()
05.     USER.Name = "Smith"
06.     USER.Gender = 0
07.     USER.Age = 28
08. End Sub
09.
```

Notice that a variable declaration of a structure must be done before a structure can be used. In this case, the variable “USER” is declared of data type structure “*structUser*”.

BLAZINGCORE COMMAND SET

BLAZINGCORE COMMAND SET

DEBUG STATEMENT

The Debug statement serves as a debugger on an embedded system. Values that are debugged using the debug statement are automatically formatted to ASCII and sent through the download port (UART2/Comm1) and displayed in the IDE's Debug Window.

`Debug.Print "[string]"` | String Variable | String Const

The Debug Statement returns Strings back to the PC. Therefore, all expressions must be in String or converted to a String (CSTR – Convert To String) before or during the instruction. (More information on the String data type is available under the String Category.)

Code:

```
01. Dim S As String
02. Dim I As Integer
03.
04. Public Sub Main()
05.     S = "This Is A Test"
06.     I = 5
07.     Debug.Print "Hello World!"
08.     Debug.Print S
09.     Debug.Print Cstr(I)
10. End Sub
11.
```

Output :

```
01. Hello World!
02. This is a test
03. 5
04.
```

As illustrated above, all Debug Statements automatically end with a carriage return (new-line). To display String continuously on the same line, we make use of the “;” semicolon.

Code:

```
01. Dim S As String
02. Dim I As Integer
03.
04. Public Sub Main()
05.     S = "The Number Is:"
06.     I = 5
07.     Debug.Print "Hello World!"
08.     Debug.Print S; " ";
09.     Debug.Print Cstr(I)
10. End Sub
11.
```

Output :

```
01. Hello World!
02. The Number Is: 5
03.
```

Of course, the Debug Statement accepts a mix of all the above. You may put the Cstr(I) after the semicolon on line 5.

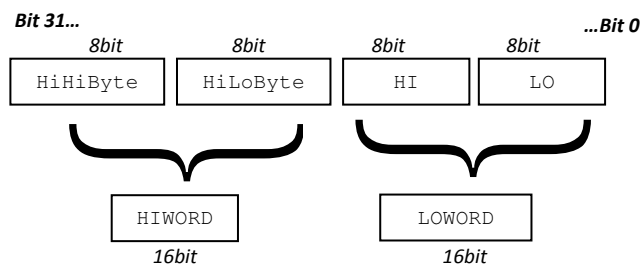
BIT, BYTE & WORD MANIPULATION

System commands are provided to enable users to manipulate values in bit, byte or word format. The following table lists the available operations and their respective description.

`<Operator>(value)`

Operator	Description
Abs	Returns the absolute value of its argument
SetBit	Set an individual bit at a specified position to logical 1. Bit Position (0 - 31) <i>Eg. SetBit(variable as integer, bit position)</i>
ClrBit	Set an individual bit at a specified position to logical 0. Bit Position (0 - 31) <i>Eg. ClrBit(variable as integer, bit position)</i>
GetBit	Get the logical state of an individual bit at a specified position.
HI	Returns the High Byte in the Low Word Portion of a 32bit Integer Value.
LO	Returns the Low Byte in the Low Word Portion of a 32bit Integer Value.
HiHiByte	Returns the High Byte in the High Word Portion of a 32bit Integer Value.
HiLoByte	Returns the Low Byte in the High Word Portion of a 32bit Integer Value.
HiWord	Returns the High Word Portion of a 32bit Integer Value.
LoWord	Returns the Low Word Portion of a 32bit Integer Value.

32bit Integer



I/O COMMANDS

`<Command>(pin no.)`

Command	Description
High	Set a specified pin's logical state to '1'.
Low	Set a specified pin's logical state to '0'.
GetPin	Returns logical state of specified pin.
GetADC	Returns converted digital values of an analog device attached to an ADC pin.

SYSTEM DELAY

DELAY

Equivalent to `DelayMS`, where time is delayed in milliseconds.

`Delay(time)`

DELAYUS

Time is delayed in microseconds.

`DelayUS(time)`

RANDOM GENERATOR

Generate Random Integer Values.

RNDSEED()

Initialise the random generator with a seed before using it to generate values.

```
OS.RndSeed(Seed)
```

RND()

Generate random integer values with a specified range.

```
OS.Rnd(Range)
```

Code:

```
01. Public Sub Main()  
02. Dim I As Integer  
03. OS.RndSeed(10)  
04. I = OS.Rnd(5)  
05. Do  
06. Debug.Print Cstr(I)  
07. Delay(200)  
08. Loop  
09. End Sub
```

Output :

Output values ranging from 0-4 will be randomly generated.

BLAZINGCORE OS COMMAND SET

OS

The following members listed under OS contain methods that allow the BCore Operating System to natively handle peripheral, memory and communication protocols of components onboard the BCore Microcontroller Board.

Members	Description
Comm1	Handles methods for serial communication on Comm1
Comm2	Handles methods for serial communication on Comm2
LED1	Provides methods to toggle the onboard LED1 on or off
LED2	Provides methods to toggle the onboard LED2 on or off
MemCard	Provides access to methods for handling an external memory card(MMC) and file operations
Memory	Provides access to methods for handling data arrays stored in memory
PB1	Returns logical state of Push Button 1 onboard the BCore Board
PB2	Returns logical state of Push Button 2 onboard the BCore Board
Reset	Commands the BlazingCore OS to perform a soft reset on itself.
Run	Commands the BlazingCore OS to run.
Timer	Provides access to system timer methods

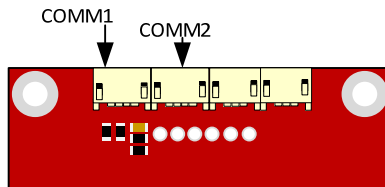
UART

Serial communication on the BCore is available through the use of UART.

The BCore board has 2 UARTs supported by the BCoreOS under the name of Comm1 and Comm2.

UART settings on the BCore are;

- Full duplex, 8bit transmission
- No parity
- One Stop Bit
- Baud Rate: 115200bps (Different baud rates will be supported in subsequent firmware upgrades)
- 4 word deep FIFO Data Receive Buffer



COMM1 / COMM2

Handles methods for serial communication on Comm1

Methods	Description
HasData	Returns state of Data Receive buffer
Rx	Returns byte data stored in Data Receive buffer
Tx	Transmit specified byte data.

.HASDATA

Returns state of Data Receive buffer

```
Variable = Comm1.HasData()
Variable = Comm2.HasData()
```

Returns;

- Logical **True** if data is present in the data receive buffer,
- Logical **False** if the buffer is empty.

.RX

Returns byte data stored in Data Receive buffer

*Data is retrieved one byte at a time.

```
Variable = Comm1.RX()
```

```
Variable = Comm2.RX()
```

.TX

Transmit specified byte data

*Transmission is done one byte at a time.

```
Comm1.TX(byte Data)
```

```
Comm2.TX(byte Data)
```

WORKING EXAMPLE

The following code is a working example of the use of the above commands to perform checking, receiving and transmission of data between the 2 COMM Ports.

Data that is received on either port is echoed to the other port.

Tests may be conducted with the help of a HyperTerminal or equivalent software.

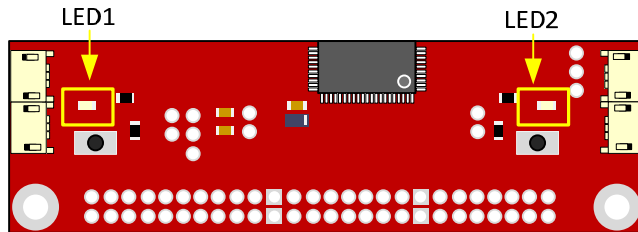
Code:

```
01. 'SIMPLE ECHO PROGRAM BETWEEN THE TWO COMM PORTS
02. Public Sub Main()
03. Dim RX as integer
04. Do
05.     If OS.Comm1.HasData() Then
06.         rx = OS.Comm1.RX()
07.         OS.Comm2.TX(rx)
08.     End If
09.     If OS.Comm2.HasData() Then
10.         rx = OS.Comm2.RX()
11.         OS.Comm1.TX(rx)
12.     End If
13. Loop
14.
15. End Sub
```

.LED1 / LED2

Provides methods to toggle the onboard LED on or off

Methods	Description
High	Turn the LED on
Low	Turn the LED off



.HIGH

Turn the LED on

```
OS.LED1.High()
OS.LED2.High()
```

.LOW

Turn the LED off

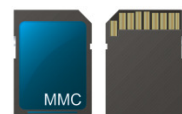
```
OS.LED1.Low()
OS.LED2.Low()
```

EXTERNAL MEMORY CARD (MMC)

BCore provides support to accessing files located in the external memory card (MMC). Bus line of the External Memory Card on the OLED Interfacing Board to the BCore Board would have lined up upon stacking. As such, no pin definitions are required on the user's part.

.MEMCARD

Provides access to methods for handling an external memory card(MMC) and file operations



Members	Description
Dir	Provides methods to access the directories within the external memory card
File	Provides methods to access files and carry out file stream operations
Init	Handles the necessary one-time initialisation of the memory card before use

DIR

Provides methods to access the directories within the external memory card

Methods	Description
Change	Change current directory within the external memory card
FindFirst	Returns first occurrence of a specified string in names of all the files in current directory
FindNext	Returns next occurrence of a specified string in names of all the files in current directory
Get	Get path of current directory

.CHANGE

Change current directory to a specified one within the external memory card

Variable = **OS.MemCard.DIR.CHANGE**(*STRING* PATH)

Parameter/s:

- Path of desired directory in *String*

Returns;

- **0**: Operation Successful
- **All other values**: Error

Code:

```

01. Public Sub Main()
02. Dim Str as String
03. Dim I as Integer
04.
05. STR = "\SDIR_2" 'FOLDER WITHIN MMC CALLED SDIR_2
06. I = OS.MEMCARD.DIR.CHANGE(STR) 'GO INTO FOLDER
07. DEBUG.PRINT "CHANGE DIRECTORY STATUS: ";CSTR(I)
08.
09. End Sub

```

.FINDFIRST

Returns first occurrence of a specified string in names of all the files in current directory

Variable = **OS.MemCard.DIR.FINDFIRST**(STR1, STR2)

Parameter/s:

- **STR1** = Specified *string* to search for
- **STR2** = *String Variable* for Command to return *File Name* in which STR1 was found.

Returns;

- **0**: File Found
- **All other values**: No matches.

META CHARACTERS

The code below is an example of how to retrieve the first found file with the specified string out of all the files available within the current directory of the external memory card.

Use of * is a meta character representing any single character or group of characters.

By using *.* , the * stands in for any file name, as well as for any file extension.

Code:

```

01. Public Sub Main()
02. Dim Str1, Str2 as String
03. Dim I as Integer
04. STR1 = "*.*" '* = Wild card /meta character
05. I = OS.MEMCARD.DIR.FINDFIRST(STR1, STR2)
06. DEBUG.PRINT "FIND FIRST ";CSTR(I); " ";STR2
07. End Sub

```

.FINDNEXT

Returns next occurrence of a specified string in names of all the files in current directory; Continuing from where the previous file was found.

IMPORTANT: Use of `OS.MEMCARD.DIR.FINDFIRST` must precedent this command

Variable = `OS.MEMCARD.DIR.FINDNEXT`(STR2)

Parameter/s:

- **STR2** = *String Variable* for Command to return *File Name* in which specified string to search for (stated in the use of command `OS.MEMCARD.DIR.FINDFIRST`) was found.

Returns;

- **0:** File Found
- **All other values:** No matches.

The code below is an example of how to retrieve the first 2 files with residing within the current directory of the external memory card.

Code:

```
01. Public Sub Main()
02. Dim Str1, Str2 as String
03. Dim I as Integer
04. STR1 = "*" "*" '*' = Wild card/ meta character
05. I = OS.MEMCARD.DIR.FINDFIRST(STR1, STR2)
06. DEBUG.PRINT " FIND FIRST ";CSTR(I); " ";STR2
07. I = OS.MEMCARD.DIR.FINDNEXT(STR2)
08. DEBUG.PRINT " FIND NEXT ";CSTR(I); " ";STR2
09. End Sub
```

.GET

Get path of current directory

Variable = `OS.MemCard.DIR.CHANGE`(STRING PATH)

Parameter/s:

- **PATH;** String variable that contains specified path within the external memory card to change directory to.

Returns;

- **0:** File Operation Successful
- **All other values:** Error

Code:

```
01. Public Sub Main()
02. Dim Str1, Str2 as String
03. Dim I as Integer
04. STR1 = "*" "\" "*" '*' = Wild card/ meta character
05. I = OS.MEMCARD.DIR.CHANGE(STR1)
06. DEBUG.PRINT " CHANGE DIR: ";CSTR(I) 'I = Success/Error
07. I = OS.MEMCARD.DIR.GET(STR2)
08. DEBUG.PRINT CSTR(I); " DIR : [" ;STR2; "]"
09. End Sub
```

FILE

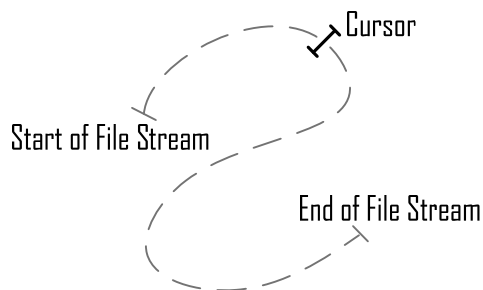
The BCore OS uses streams to support reading from and writing to files located in the external memory card (MMC).

Methods	Description
Close	Closes a currently opened file stream
GetPos	Returns current position of data being read within an open file
IsEOF	Checks if end of file stream has been reached
Open	Open a specified file
Read	Start reading a specified number of bytes of data from the file stream
Rewind	Sets cursor position back to the beginning of the file stream
Seek	Sets cursor position to a specified index within the file stream
Write	Perform write operations to specified file stream

FILE STREAMS

When the BCore OS accesses the files located in the external memory card (MMC), it performs file operations through the use of streams. A file stream is essentially a single dimensional set of contiguous data obtained from the file. A file stream has a start and an end, and a cursor that represents the current position of data being read from the file.

A visual representation of a file stream can be seen in the image below.



.CLOSE

Closes a currently opened file stream

Variable = **OS.MemCard.File.Close**(*fileNum*)

Parameter/s:

- **fileNum**; File Number to close.

Returns;

- **0**: File Operation Successful
- **All other values**: Error

The following code is an example of how to initialise the external memory card, open a file and assign a number to it, and closing the file using the previously assigned number.

Code:

```

01. Public Sub Main()
02. Dim Str1, Str2 as String
03. Dim I as Integer
04. I = OS.MEMCARD.INIT()
05. DEBUG.PRINT "INIT ";CSTR(I)
06. STR2 = "FILE_01.TXT"
07. I = OS.MEMCARD.FILE.OPEN(1, STR2, 0)
08. DEBUG.PRINT "OPEN ";CSTR(I)
09. I = OS.MEMCARD.FILE.CLOSE(1)
10. DEBUG.PRINT "CLOSE ";CSTR(I)
11.
12. End Sub

```

.GETPOS

Returns current position of cursor in a specified file stream

Variable = **OS.MemCard.File.GetPos** (fileNum)

Parameter/s:

- **fileNum**; File Number in which to get the cursor position.

Returns;

- **Cursor Position**

Code:

```
01. Public Sub Main()
02. Dim Str1, Str2 as String
03. Dim I as Integer
04. I = OS.MEMCARD.INIT()
05. DEBUG.PRINT "INIT ";CSTR(I)
06. STR2 = "FILE_01.TXT"
07. I = OS.MEMCARD.FILE.OPEN(1, STR2, 0)
08. DEBUG.PRINT "OPEN ";CSTR(I)
09. I = OS.MEMCARD.FILE.GETPOS(1)
10. DEBUG.PRINT "GETPOS ";CSTR(I)
11.
12. End Sub
```

.ISEOF

Checks if cursor has reached the end of the specified file stream

Variable = **OS.MemCard.File.IsEOF** (fileNum)

Parameter/s:

- **fileNum**; File Number in which to check the cursor position.

Returns:

- Logical **True** if cursor is at the end of the file stream;
- Logical **False** if cursor is not at the end of the file stream.

.OPEN

Open a specified file located in the external memory card (MMC).

Variable = **OS.MemCard.File.Open** (FileNum, FileName, R/W)

Parameter/s:

- **FileNum**; Assign an available number to associate the file with
- **FileName**; String variable containing name of file to open
- **R/W**;
 - **0**, to open a file for **read** operations,
 - **1**, to open the file for **write** operations
 - **2**, to open the file for **read and write** operations

Returns:

- **0**: File successfully opened
- **All other numbers**: Error

Code:

```
01. Dim Str1, Str2 as String
02. Dim I as Integer
03. I = OS.MEMCARD.INIT()
04. DEBUG.PRINT "INIT ";CSTR(I) 'I = Success/Error
05. STR2 = "FILE_01.TXT"
06. 'Assign number 1 to FILE_01.TXT and open as read-only
07. I = OS.MEMCARD.FILE.OPEN(1, STR2, 0)
08. DEBUG.PRINT "FILE OPEN STATUS: ";CSTR(I) 'I = Success/Error
```

.READ

Start reading a specified number of bytes of data from the file stream

Variable = **OS.MemCard.File.GetPos**(fileNum, StrVar, BytesToRead)

Parameter/s:

- **fileNum**; File Number in which to get the cursor position.
- **StrVar**; String variable to read the bytes into
- **BytesToRead**; Specified number of bytes to read into StrVar

Returns;

- **0**: Read Successful
- **All other numbers**: Error

Code:

```
01. Public Sub Main()
02. Dim Str1, Str2 as String
03. Dim I as Integer
04. I = OS.MEMCARD.INIT()
05. DEBUG.PRINT "INIT ";CSTR(I)
06. STR1 = "FILE_01.TXT"
07. I = OS.MEMCARD.FILE.OPEN(1, STR1, 0)
08. DEBUG.PRINT "OPEN ";CSTR(I)
09. I = OS.MEMCARD.FILE.READ(1, STR2, 10)
10. DEBUG.PRINT "String Read: "; CSTR(I); " <";STR2;">"
11. End Sub
```

.REWIND

Sets cursor position back to the beginning of the file stream

Variable = **OS.MemCard.File.Rewind**(fileNum)

Parameter/s:

- **fileNum**; File Number in which to set the cursor position back to the beginning of the file stream.

Returns:

- **0**: Read Successful
- **All other numbers**: Error

.SEEK

Sets cursor position to a specified index within the file stream

Variable = **OS.MemCard.File.Seek**(FileNum, Index)

Parameter/s:

- **FileNum**; Assign an available number to associate the file with
- **Index**: Position within the file stream to set the cursor

Returns:

- **0**: File successfully opened
- **All other numbers**: Error

Code:

```
01. Dim I as Integer
02. I = OS.MEMCARD.INIT()
03. I = OS.MEMCARD.FILE.OPEN(1, STR1, 0)
04. I = OS.MEMCARD.FILE.SEEK(1, 5) 'Set Cursor Position to 5
05. DEBUG.PRINT "SEEK 5 ";CSTR(I)
06. I = OS.MEMCARD.FILE.GETPOS(1)
07. DEBUG.PRINT "GETPOS ";CSTR(I) 'I = 5
```

.WRITE

Perform write operations to specified file stream

Variable = **OS.MemCard.File.Write**(fileNum, StrVar, BytesToWrite)

Parameter/s:

- **fileNum**; File Number in which to get the cursor position.
- **StrVar**; String variable containing characters to write to the file.
- **BytesToWrite**; Number of bytes from StrVar to write to the file.

Returns;

- **0**: Write Successful
- **All other numbers**: Error

Code:

```
01. Public Sub Main()  
02. Dim Str1, Str2 as String  
03. Dim I as Integer  
04. I = OS.MEMCARD.INIT()  
05. DEBUG.PRINT "INIT ";CSTR(I)  
06. STR1 = "FILE_NEW.TXT"  
07. I = OS.MEMCARD.FILE.OPEN(1, STR1, 2) 'Open as read & write  
08. DEBUG.PRINT "OPEN ";CSTR(I) 'I = Success/Error  
09. STR2 = "12345"  
10. I = OS.MEMCARD.FILE.WRITE(1, STR2, 3) 'write "123" to file  
11. DEBUG.PRINT "WRITE ";CSTR(I) 'I = Success/Error  
12. End Sub
```

INIT

Handles the necessary one-time initialisation of the memory card before it can be used

Variable = **OS.MemCard.Init**()

Returns;

- **0**: External Memory Card Successfully Initialised
- **All other numbers**: Error

Code:

```
01. Public Sub Main()  
02. I = OS.MEMCARD.INIT()  
03. DEBUG.PRINT "INIT ";CSTR(I) 'I = Success/Error  
04. End Sub
```

MEMORY

Provides access to methods for handling data arrays stored in memory

Methods	Description
Copy	Copies elements from one array to another
CopyExt	Copies a specified range of elements from one array to another
FindFirst	Min Returns index of first occurrence of the smallest number in a specified range of elements
	Max Returns index of first occurrence of the largest number in a specified range of elements

.COPY

Copies elements from one array to another.

OS.Memory.Copy(Source, Destination, Length)

Parameter/s:

- **Source;** Variable Array to copy from
- **Destination;** Variable Array to copy to
- **Length;** Number of elements to copy over

Code:

```

01. Dim Arr1(20) as Integer
02. Dim Arr2(20) as Integer
03. Public Sub Main()
04. Dim I as Integer
05. 'Initialise the Array with some values
06. For I = 0 To 5
07.     Arr1(I) = I
08. Next
09. OS.Memory.Copy(Arr1, Arr2, 3)
10. For I = 0 To 5
11.     DEBUG.PRINT CSTR(Arr2(I)); ", ";
12. Next
13. End Sub
    
```

Output :

```

01. 0, 1, 2, 0, 0, 0,
    
```

.COPYEXT

Copies elements from one array to another.

OS.Memory.CopyExt(Source, Source Start, Destination, Destination Start, Length)

Parameter/s:

- **Source;** Variable Array to copy from
- **Source Start;** Index of Source to start copying elements from
- **Destination;** Variable Array to copy to
- **Destination Start;** Index of Destination to start copying elements to
- **Length;** Number of elements to copy over

Code:

```

01. Dim Arr1(20) as Integer
02. Dim Arr2(20) as Integer
03. Public Sub Main()
04. Dim I as Integer
05. 'Initialise the Array with some values
06. For I = 0 To 5
07.     Arr1(I) = I
08. Next
09. OS.Memory.CopyExt (Arr1, 2, Arr2, 1, 3)
10. For I = 0 To 5
11.     DEBUG.PRINT CSTR(Arr2(I));", ";
12. Next
13.
14. End Sub

```

Output :

```
01. 0, 2, 3, 4, 0, 0,
```

.FINDFIRST

Performs a search function into specified data arrays

Methods	Description
Min	Returns index of first occurrence of the smallest number in a specified range of elements
Max	Returns index of first occurrence of the largest number in a specified range of elements

.MIN

Returns index of first occurrence of the smallest number in a specified range of elements

Variable = **OS.Memory.FindFirst.Min**(*Memory*, *Start*, *Length*)

Parameter/s:

- **Memory;** Data array to carry out the search
- **Start;** Index of Data Array to start searching from
- **Length;** Number of elements to search

Returns;

- **Index** of element at which the condition was satisfied

Code:

```

01. Dim Arr1(20) as Integer
02. Dim Arr2(20) as Integer
03. Public Sub Main()
04. Dim I as Integer
05. 'Initialise the Array with some values
06. For I = 0 To 5
07.     Arr1(I) = I
08. Next
09. I = OS.MEMORY.FINDFIRST.Min (ARR1, 0, 10)
10. debug.print "pos = ";cstr(I)
11.
12. End Sub

```

Output :

```
01. pos = 0
```

.MAX

Returns index of first occurrence of the largest number in a specified range of elements

Variable = `OS.Memory.FindFirst.Max`(Memory, Start, Length)

Parameter/s:

- **Memory;** Data array to carry out the search
- **Start;** Index of Data Array to start searching from
- **Length;** Number of elements to search

Returns;

- **Index** of element at which the condition was satisfied

Code:

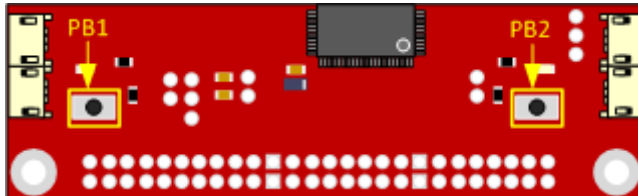
```
01. Dim Arr1(20) as Integer
02. Dim Arr2(20) as Integer
03. Public Sub Main()
04. Dim I as Integer
05. 'Initialise the Array with some values
06. For I = 0 To 5
07.     Arr1(I) = I
08. Next
09. I = OS.MEMORY.FINDFIRST.Max(Arr1, 0, 10)
10. debug.print "pos = "; cstr(I)
11.
12. End Sub
```

Output :

```
01. pos = 5
```

.PB1/PB2

Returns logical state of onboard push buttons



Variable = `OS.PB1`()

Variable = `OS.PB2`()

Returns;

- **0;** Button is not pressed
- **1;** Button is pressed

The following program polls both the onboard buttons for its logical state 5 times a second to detect if any of the buttons are pressed.

Code:

```
01. Dim BUTTON1, BUTTON2 as Integer
02. Do
03.     BUTTON1 = OS.PB1()
04.     BUTTON2 = OS.PB2()
05.     Debug.Print Cstr(BUTTON1); " "; Cstr(BUTTON2)
06.     Delay(200)
07. Loop
```

.RESET

Commands the BlazingCore OS to perform a soft reset on itself.

`OS.Reset()`

.RUN

Commands the BlazingCore OS to run.

`Variable = OS.Run()`

.TIMER

Provides access to control the chip's system timer.

Member	Description
<code>Init()</code>	Initialise OS Timer with a specified prescale value index.
<code>Read</code>	Returns time elapsed in terms of clock cycles (dependent on prescaler)
<code>Set</code>	Set Timer Value
<code>Start</code>	Start the Timer
<code>Stop</code>	Stop the Timer

.INIT

Initialise OS Timer with a specified prescale value index.

`OS.TIMER.INIT(prescale index)`

Prescale Index	Prescale Factor
1	1 (default)
2	2
3	4
4	8
5	16

.READ

Returns time elapsed in terms of clock cycles (dependent on prescaler)

`Variable = OS.TIMER.Read()`

.SET

Set Timer Value to start from

`OS.TIMER.Set(value)`

.START

Start the Timer

`OS.TIMER.Start()`

.STOP

Stop the Timer

`OS.TIMER.Stop()`

TIMER WORKING EXAMPLE

The following is an example code on using the system timer with a prescaler set to a factor of 2.

Code:

```
01. Dim TIME as Long
02. Public Sub Main()
03. OS.TIMER.INIT(2)
04. OS.TIMER.Set(0)
05. OS.TIMER.Start()
06. TIME = OS.TIMER.Read()
07. OS.TIMER.Stop()
08.
09. Debug.Print "CYCLES ELAPSED: ";Cstr(TIME)
10.
11. End Sub
12.
```

BLAZINGCORE I/O COMMAND SET

I/O

The following table lists the commands available for addressing I/O ports, as well as methods for getting information from a particular I/O pin.

Operator	Description
GetADC	Returns converted digital values of an analog device attached to an ADC pin.
GetPin	Returns logical state of specified pin.
High	Set a specified pin's logical state to '1'.
Low	Set a specified pin's logical state to '0'.
Port	Address all pins in specified port
PortA	Address all pins in PortA (5bit)
PortB	Address all pins in PortB (16bit)
PortD	Address all pins in PortD (16bit)
PortE	Address all pins in PortE (8bit)

Note: Using commands `GetADC` and `GetPin` on its own is also allowed.

*Refer to I/O Commands under BCore OS Command Reference.

GETADC

Returns converted digital values of an analog device attached to an ADC pin. (ADC Pins; 17 to 32)

Integer variable = IO.GetADC(pin)

Code:

```

01. Dim I As Integer
02.
03. Public Sub Main()
04. Do
05.     I = IO.GetADC(17)
06.     Debug.Print Cstr(I)
07.     Delay(200)
08. Loop
09. End Sub

```

GETPIN

Returns logical state (High/Low) of specified pin.

Integer variable = IO.GetPin(pin)

The following example code polls pin 61 (Button1 on the BCore Board) at a rate of 5 times a second to detect if the push button has been pressed.

Code:

```

01. Dim I As Integer
02.
03. Public Sub Main()
04. Do
05.     I = IO.GetPin(61)
06.     Debug.Print Cstr(I)
07.     Delay(200)
08. Loop
09. End Sub

```

I/O COMMAND REFERENCE

HIGH

Set a specified pin's logical state to '1'.

```
IO.High(pin)
```

LOW

Set a specified pin's logical state to '0'.

```
IO.Low(pin)
```

The following code is an example program that illustrates the use of the commands High and Low to toggle pin 60 (LED1 onboard the BCore Board), resulting in a blinking LED program. Please note, both codes listed below are equivalent.

Code:

```
01. Public Sub Main()  
02.  
03. Do  
04.     IO.High(60)  
05.     Delay(100)  
06.     IO.Low(60)  
07.     Delay(100)  
08. Loop  
09. End Sub
```

Code:

```
01. Public Sub Main()  
02.  
03. Do  
04.     OS.LED1.High()  
05.     Delay(100)  
06.     OS.LED1.Low()  
07.     Delay(100)  
08. Loop  
09. End Sub
```

PORT

Address all pins in specified port

*Please refer to the I/O Pinout Description table for list of pins and their respective ports.

```
IO.Port(port No.)
```

Set: `IO.Port(port No.) = value`

Get: `value = IO.Port(port No.)`

ADDRESSABLE PORT REGISTERS

The following ports are addressable with pins corresponding to the respective port registers of the PIC32 Chip.

*Please refer to the I/O Pinout Description table for list of pins and their ports with respect to the port registers.

PORTA

```
IO.PortA
```

PORTA 5bit	RA0	RA1	RA2	RA3	RA4
Bcore Pins	49	50	51	52	53

PORTB

```
IO.PortB
```

PORTB 16bit	RB0	RB1	RB2	RB3	RB4	RB5	RB6	RB7	RB8	RB9	RB10	RB11	RB12	RB13	RB14	RB15
Bcore Pins	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17

PORTD

```
IO.PortD
```

PORTD 16bit	RD0	RD1	RD2	RD3	RD4	RD5	RD6	RD7	RD8	RD9	RD10	RD11	RD12	RD13	RD14	RD15
Bcore Pins	40	39	38	37	36	35	34	33	41	42	43	44	45	46	47	48

PORTE

PORTE 8bit	RE0	RE1	RE2	RE3	RE4	RE5	RE6	RE7
Bcore Pins	9	10	11	12	13	14	15	16

```
IO.PortE
```

















BLAZINGCORE OLED COMMAND REFERENCE

OLED COMMAND REFERENCE

The following pages list the members in the library available to interface with the OLED Display. Ready methods like printing text to screen, drawing primitives or even placing bitmaps to the screen are available for immediate usage. Examples of use are also included.

OLED

Part of the BCoreIDE library consists of optimized methods to control the OLED natively to ensure efficiency and speed of data transfer to the OLED Driver.

Method	Description
 Accel	Provides Access to reading values from the onboard accelerometer
 Background	Set Background colour
 Clear()	Clears the screen
 Cursor	Get or Set the Cursor Position
 Draw	Graphic library for drawing Primitives and Bitmaps to Screen <i>*refer to subtopic for details.</i>
 FontType	Set the Font Size
 Foreground	Set Foreground colour
 GRAM	Provides Read-Only Access to the Graphics RAM
 Init()	Does the necessary one-time initialization of the OLED with a specified colour
 LDR	Provides access to reading values from the onboard Light Dependent Resistor (LDR)
 Orientation	Set the orientation of the screen to portrait or landscape
 Print	Print text to OLED
 ReadRegister	Provides read abilities to the OLED driver (S6E63D6) registers ¹
 SetColour	Set foreground and background colours
 Touch	Provides access to reading values from the touch screen
 WriteRegister	Provides write abilities to the OLED driver (S6E63D6) registers ¹

¹ Please refer to S6E63D6 Driver IC Datasheet for full register listing.

.ACCEL

Provides Access to methods for reading values from the onboard accelerometer.

Member	Description
X	Get X axis reading
Y	Get Y axis reading
Z	Get Z axis reading

```
<variable> = OLED.Accel.X()
```

Code:

```
01. Public X,Y,Z as Integer
02. Public Sub Main()
03.
04.     Do
05.         X = OLED.Accel.X()
06.         Y = OLED.Accel.Y()
07.         Z = OLED.Accel.Z()
08.         Debug.Print Cstr(X); " ";Cstr(Y); " ";Cstr(Z)
09.         Delay(200)
10.     Loop
11.
12. End Sub
```

OLED COMMAND REFERENCE

.BACKGROUND

Set the background colour of the OLED.

`OLED.Background.COLOUR`

Code:

```
01. 'CODE1
02. Public Sub Main()
03. 'INIT THE OLED AND SET THE BACKGROUND COLOUR TO BLACK
04. OLED.Init(BCK.COLOUR.BLACK)
05. 'CHANGE THE BACKGROUND COLOUR TO FUSCHIA
06. OLED.Background.Colour = BCK.COLOUR.Fuschia
07. End Sub
```

.CLEAR()

Clears the screen.

`OLED.Clear()`

.CURSOR

Get or Set the Cursor Position

Member	Description
X	Set X coordinate of cursor
Y	Set Y coordinate of cursor
Home	Sets the cursor to (0,0) position

To set the cursor position;

```
OLED.Cursor.X = value
OLED.Cursor.Y = value
OLED.Cursor.Home()
```

Value: Any literal, constant, or expression

To get the cursor position;

```
value = OLED.Cursor.X
value = OLED.Cursor.Y
```

.DRAW

Uses inbuilt native libraries in BCore to draw primitives and bitmaps on to the screen.

All primitives are drawn using the specified foreground colour.

Member	Description
Point	Plots a pixel at a specified point on the OLED Screen
Points	Plots an array of points.
Line	Draws a Line.
Lines	Draws an array of Lines.
Rectangle	Draws a Rectangle of specified sizing at a specified position.
FillRectangle	Draws a Filled Rectangle of specified sizing at a specified position.
Circle	Draws a Circle of specified sizing at a specified position.
BitmapFromDFlash	Draws a bitmap file from the data flash memory chip.
BitmapFromMemCard	Draws a bitmap from External Memory Card (MMC) to the Screen

OLED COMMAND REFERENCE

.POINT

Plots a single pixel at a specified point on the OLED Screen.

```
OLED.Draw.Point (point)
```

.POINTS

Plots an array of pixels.

Parameters:

- *Points()* = point array;
- *Zero-offset start* = zero-offset start index of point array to start plotting from;
- *Number of points* = number of points to plot.

```
OLED.Draw.Points (points(), zero-offset start, number of points)
```

.LINE

Draws a Line from *point1* to *point2* on the OLED Screen.

```
OLED.Draw.Line (point1, point2)
```

.LINES

Draws an array of points as a continuous connected line.

Parameters:

- *Points()* = point array;
- *Zero-offset start* = zero-offset start index of point array to start plotting from;
- *Number of points* = number of points to plot.

```
OLED.Draw.Lines (points(), zero-offset start, number of points)
```

.RECTANGLE

Draws a Rectangle using 2 points.

```
OLED.Draw.Rectangle (point1, point2)
```

.FILLRECTANGLE

Draws a solid/filled rectangle of foreground colour from *point1* to *point2* on the OLED Screen.

```
OLED.Draw.FillRectangle (point1, point2)
```

.CIRCLE

Draws a circle. Parameters accept center point position and radius of circle.

```
OLED.Draw.Circle (center-point, radius)
```

Point1 <x1, y1>

Points(1) <x1, y1>
Points(2) <x2, y2>
Points(0) <x0, y0>
Points(3) <x3, y3>

Point2 <x2, y2>
Point1 <x1, y1>

Points(1) <x1, y1>
Points(2) <x2, y2>
Points(0) <x0, y0>
Points(3) <x3, y3>

Point1 <x1, y1>
Point2 <x2, y2>

Point1 <x1, y1>
Point2 <x2, y2>

Center-point
<x,y>

DRAWING BITMAPS TO SCREEN

Storage and Rendering of Bitmap files are supported by the BCore OS.

However, due to the nature of bitmap files, they tend to require a large amount of memory. As such, Bitmaps are only allowed to be stored on External Memory Cards (like MMC Cards), or the external DataFlash chip.

Depending on where you store your Bitmap Files, a single line command is able to fetch the image file and draw it onto the OLED screen, provided it is given the correct information.

Please note: The commands to draw bitmaps to screen returns a value and must be assigned to a variable.

This value will indicate if there has been any error during file fetching and/or drawing to the screen.

Value '0': Successful

Value '1': Error

Note: ¹Bitmaps must be in Windows® Bitmap .bmp format.

²Bitmaps of bit depth 16bit 565 Mode, 24bpp, 32bpp are supported.

³For applications that require the OLED to update images really quickly, we recommend that the bitmaps are first converted to the 16bit 565 RGB Format before importing it into the storage media for the BCore to access, since 24bpp and 32bpp files are converted by the chip to the 16bit 565 format before sending to the OLED Display, thereby imposing a limitation to the update rate from the BCore to the OLED Display.

.BITMAPFROMDFLASH

Draws a bitmap on the OLED screen from a .bmp file stored in the external DataFlash chip.

Code:

```
01. Dim P1 As Point
02. Public Sub Main()
03. Dim I1 As Integer
04. DEBUG.PRINT "OLED - BITMAP"
05. DELAY(200)
06. OLED.INIT(0)
07. OLED.SETCOLOUR.FOREGROUND(31)
08. I1 = ADDRESSOF(DATA1.B_PLAYER)
09. P1.X = 50
10. P1.Y = 20
11. OLED.DRAW.BitmapFromDFlash(P1, I1)
12. End Sub
```

Note: Please refer to **APPENDIX A** for more details on transferring Bitmap Files from the PC to the BCore's External DataFlash.

.BITMAPFROMMEMCARD

Draws a bitmap on the OLED screen from a .bmp file stored in the External Memory Card (MMC / SD Compatible).

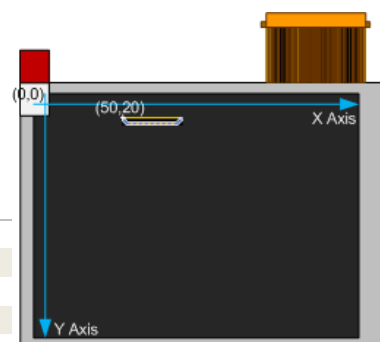
`OLED.Draw.BitmapFromMemCard(FileName,Position)`

Parameters:

- **FileName:** Name of Bitmap File in *String*
- **Position:** *Point* XY location to draw bitmap

Code:


```
01. Dim P1 As Point
02. Public Sub Main()
03. Dim I As Integer
04. Dim STR1 AS STRING
05. OLED.INIT(0)
06. OLED.SETCOLOUR.FOREGROUND(31)
07. I = OS.MEMCARD.INIT()
08. STR1 = "B_PLAYER.BMP"
09. P1.X = 50
10. P1.Y = 20
11. I = OLED.DRAW.BITMAPFROMMEMCARD(STR1, P1)
12. End Sub
```



OLED COMMAND REFERENCE

.FONTTYPE

Sets the font size of text printed to the screen.

Available font sizes are 8x8, 10x12, 16x16 pixels 

`OLED.FontType`

Assignment

- `OLED.FontType = font size;`
 - `0 = 8x8 pixel;`
 - `1 = 10x12 pixel;`
 - `2 = 16x16 pixel`

A BCore Constant that stores the font size values is available to make the code more readable.

The following shows a code comparison on how to set the font size using direct assignment of values and using the BCore Constants. Both codes achieve the same effect.

Code:

```
01. OLED.FontType = 0
02. OLED.FontType = 1
03. OLED.FontType = 2
```

Code:

```
01. OLED.FontType = BCK.Font.Small
02. OLED.FontType = BCK.Font.Medium
03. OLED.FontType = BCK.Font.Large
```

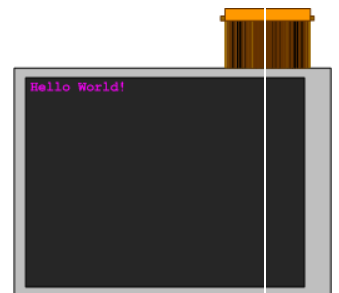
.FOREGROUND

Set the foreground colour of the OLED.

`OLED.Foreground.Colour`

Code:

```
01. 'CODE1
02. Public Sub Main()
03. 'INIT THE OLED AND SET THE BACKGROUND COLOUR TO BLACK
04. OLED.Init(BCK.COLOUR.BLACK)
05. OLED.Foreground.Colour = BCK.COLOUR.Fuschia
06. OLED.Print "Hello World!"
07. End Sub
```



OLED COMMAND REFERENCE

.GRAM

The pixels displayed on the OLED screen correspond to the image data stored in the Graphics RAM (GRAM).

Read access to the OLED GRAM is provided to the user.

Read the colour data direct from GRAM, by providing the position to read.

Read command returns the colour in 16bit 565 RGB Integer format.

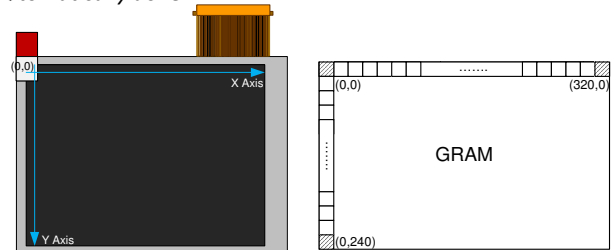
Member	Submember	Description
Read	Point	Returns the colour value from a single point in the GRAM
	Rectangle	Returns the colour values from a specified rectangular area in the GRAM into a specified user declared integer array.

```
OLED.GRAM.Read.Point(point)
```

```
OLED.GRAM.Read.Rectangle(memoryArr(),point1,point2)
```

Parameter/s:

- `memoryArr();`
 - User declared single or 2 dimensional integer array to store the colour values read from the GRAM
 - Users are expected to ensure that the memory allocation of the Integer Array is sufficient to store the total number of points being read to it
 - Data from specified area will be stored in a continuous stream inside the integer array. Any row alignments (applicable only to 2 dimensional arrays) will be automatically done.
- `point1`
 - Starting point of Rectangle
- `point2`
 - Ending point of Rectangle



.INIT()

Does the necessary one-time initialising required for the OLED to be controlled thereafter. Parameter accepts colour for the OLED screen to be initialised with.

***Note:** Colour listing available. Please refer to the page on using colour constants.

```
OLED.Init(colour)
```

Code:

```
01. 'CODE1
02. Public Sub Main()
03. 'INIT THE OLED AND SET THE BACKGROUND COLOUR TO BLACK
04. OLED.Init(BCK.COLOUR.BLACK)
05. End Sub
```

.LDR

Provides access to reading values from the onboard Light Dependent Resistor (LDR)

LDR 

```
OLED.LDR.Read()
```

Code:

```
01. Public LDR as Integer
02. Public Sub Main()
03. Do
04. LDR = OLED.LDR.READ()
05. Debug.Print Cstr(LDR)
06. Delay(200)
07. Loop
08. End Sub
```

OLED COMMAND REFERENCE

.ORIENTATION

Set the orientation of the OLED screen to portrait or landscape.

```
OLED.Orientation(Orientation)
```

Parameter/s:

- *Orientation*;
 - 0: Landscape (Default)
 - 1: Portrait



.PRINT

Prints text to screen using specified foreground colour. Works just like `Debug.Print`.

***Note:** Default text size is 8x8pixels.

```
OLED.Print "[string]" | String Variable | String Const
```

*Full set of 256 ASCII characters supported.

.READREGISTER

Returns the contents of the OLED driver (S6E63D6) register at a specified address.

```
OLED.ReadRegister(Address)
```

.WRITEREGISTER

Provides write abilities to the OLED driver (S6E63D6) registers

¹ Please refer to S6E63D6 Driver IC Datasheet for full register listing.

```
OLED.WriteRegister(Address, Value)
```

.SETCOLOUR

Set foreground and background colours

Member	Description
Background	Set Background Colour
Foreground	Set Foreground Colour

```
OLED.SetColour.Background = <colour>
```

```
OLED.SetColour.Foreground = <colour>
```

<Colour>:

*Refer to BCore Constants (BCK) for list of preset colours.

.TOUCH

Provides access to touch screen methods.

Member	Description
X	Returns optimized X coordinate of touch
Y	Returns optimized Y coordinate of touch
Raw	X Returns Raw Value of X
	Y Returns Raw Value of Y

X

Returns calculated X coordinate of a touch on the touch screen.

`OLED.Touch.X()`

Y

Returns calculated Y coordinate of a touch on the touch screen.

`OLED.Touch.Y()`

Code:

```

01. Public X,Y as Integer
02. Public Sub Main()
03.
04.
05.     Do
06.         X = OLED.Touch.X()
07.         Y = OLED.Touch.Y()
08.         Debug.Print Cstr(X); " ";Cstr(Y)
09.         Delay(200)
10.     Loop
11. End Sub
    
```

X.RAW

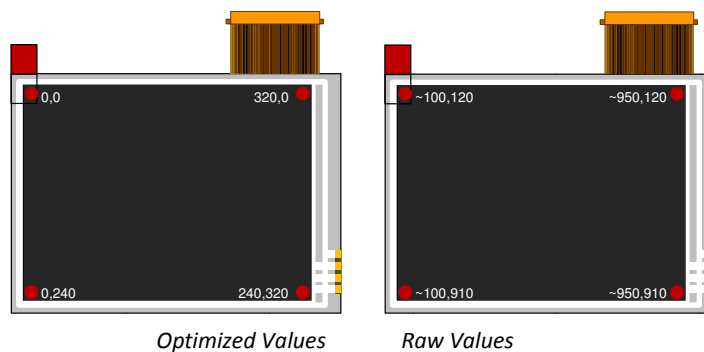
Returns raw X coordinate of a touch on the touch screen.

`OLED.Touch.X.Raw()`

Y.RAW

Returns raw Y coordinate of a touch on the touch screen.

`OLED.Touch.Y.Raw()`





BLAZINGCORE CONSTANTS

BLAZINGCORE CONSTANTS



BCK

BCore constants are constants that hold values meant to be used in conjunction with certain method and property types within the BCore command set. Currently, it holds values for use mainly with the OLED library.

	Method	Description
	Colour	Contains list of colours that represent valid 16bit (5/6/5) RGB values.
	Font	Contains values to set font sizing

COLOUR/COLOR

Contains list of colours that represent valid 16bit (5/6/5) RGB values.

Member	
	Aqua
	Black
	Blue
	Fuschia
	Grey
	Green
	Lime
	Maroon
	Navy
	Olive
	Purple
	Red
	Silver
	Teal
	White
	Yellow

```
BCK.Colour.<colour>
```

```
BCK.Color.<color>
```

Note:




- Where “BCK” represents “BlazingCore Constants”
- Each member of “Colour” contains a constant value representing a 16bit 565 mode RGB colour.
- Naming conventions “Colour” and “Color”, “Grey” and “Gray” are both supported.

FONT

Contains values to set font sizing.

Available font sizes are 8x8, 10x12, 16x16 pixels



	Size	Description
	Small	8x8
	Medium	10x12
	Large	16x16

```
BCK.Font.<size>
```

BLAZINGCORE VISION SYSTEM CAMERA COMMANDS

VISION SYSTEM CAMERA COMMANDS

CAM

Provides access to the CMOS c3038 camera module.

Member	Description
GrabFrame	Captures a single digital frame from the camera module and stores it into the GRAM of the OLED
Position	Position to display the captured frame on the OLED
ReadReg	Provides read access to the registers of the camera module
WriteReg	Provides write access to the registers of the camera module

The Camera Module must be initialised before any frame grabbing can be done.

Using the camera library built onboard the BCore OS, users are given access to the OV6630 Chip Registers for settings to be made. In the code examples that follow, 3 registers are accessed in OV6630 to configure settings for the initialising stage; *Clock Rate Control*, *Common Control A*, and *Common Control B*.

TABLE DESCRIPTION OF OV6630 I²C REGISTER SET *

*For more information please refer to OV6630 Datasheet: [URL](#)

Sub-Address (HEX)	Register	Default (HEX)	Read/Write	Description
11	CLKRC	00	RW	Clock Rate Control CLKRC[7:6] – Sync output polarity selection “00” – HSYNC=Neg, CHSYNC=Neg, VSYNC=Pos “01” – HSYNC=Neg, CHSYNC=Neg, VSYNC=Neg “10” – HSYNC=Pos, CHSYNC=Neg, VSYNC=Pos “11” – HSYNC=Pos, CHSYNC=Pos, VSYNC=Pos CLKRC[5:0] – Clock pre-scaler $CLK = (MAIN_CLOCK / ((CLKRC[5:0] + 1) \times 2)) / n$ Where n=1 if register [3E], COMO<7> is set to “1” and n=2 otherwise.
12	COMA	24	RW	Common Control A COMA[7] – SRST, “1” initiates soft reset. All registers are set to default values and chip is reset to known state and resumes normal operation. This bit is automatically cleared after reset. COMA[6] – MIRR, “1” selects mirror image COMA[5] – AGCEN, “1” enables AGC, COMA[4] – Digital output format, “1” selects 8-bit: Y U Y V Y U Y V COMA[3] – Select video data output: “1” - select RGB, “0” - select YCrCb COMA[2] – Auto white balance “1” - Enable AWB, “0” - Disable AWB COMA[1] – Color bar test pattern: “1” - Enable color bar test pattern COMA[0] – Reserved
13	COMB	01	RW	Common Control B COMB[7:6] – Reserved COMB[5] - Select data format. “1” - select 8-bit format, Y/CrCb and RGB is multiplexed to 8-bit Y bus, UV bus is tri-stated, “0” - select 16-bit format COMB[4] – “1” - enable digital output in ITU-656 format COMB[3] – CHSYNC output. “1” - horizontal sync, “0” - composite sync COMB[2] – “1” – Tri-state Y and UV busses. “0” - enable both busses COMB[1] – “1” - Initiate single frame transfer. COMB[0] – “1” - Enable auto adjust mode.

INITIALIZING THE CAMERA MODULE

Code:

```
01. PUBLIC SUB MAIN()  
02. 'Init Camera Module  
03. CAM.WRITEREG(18, 128) 'Common Control A 0x12  
04. DELAY(200)  
05. CAM.WRITEREG(18, 64+32+16+8+4)  
06. DELAY(200)  
07. CAM.WRITEREG(17, 6) 'SPEED - DONT CHANGE 0x11  
08. CAM.WRITEREG(19, 32+1) 'Common Control B 0x13  
09.  
10. END SUB
```

VISION SYSTEM CAMERA COMMANDS

GRABFRAME

Grabs a single frame from the camera module and writes it directly to the OLED's GRAM for display.

`CAM.GRABFRAME()`

Code:

```
01. PUBLIC SUB MAIN()  
02. 'Init Camera Module  
03. CAM.WRITEREG(18, 128) 'Common Control A 0x12  
04. DELAY(200)  
05. CAM.WRITEREG(18, 64+32+16+8+4)  
06. DELAY(200)  
07. CAM.WRITEREG(17, 6) 'SPEED - DONT CHANGE 0x11  
08. CAM.WRITEREG(19, 32+1) 'Common Control B 0x13  
09. DO  
10. CAM.GRABFRAME()  
11. LOOP  
12. END SUB
```

POSITION

Sets the position to display the frame from the camera module on the OLED.

`CAM.POSITION(x, y)`

Parameter/s:

- *X; Integer x-coordinate*
- *Y: Integer y-coordinate*

Code:

```
01. PUBLIC SUB MAIN()  
02. 'Init Camera Module  
03. CAM.WRITEREG(18, 128) 'Common Control A 0x12  
04. DELAY(200)  
05. CAM.WRITEREG(18, 64+32+16+8+4)  
06. DELAY(200)  
07. CAM.WRITEREG(17, 6) 'SPEED - DONT CHANGE 0x11  
08. CAM.WRITEREG(19, 32+1) 'Common Control B 0x13  
09. CAM.POSITION(40, 5) 'Set position to display frame  
10. DO  
11. CAM.GRABFRAME()  
12. LOOP  
13. END SUB
```

.READREG

Returns contents of OV6630 register at a specified address.

Variable = `CAM.READREG(Register Address)`

.WRITEREG

Write to Register.

`CAM.WRITEREG(Register Address, Value)`

Code:

```
01. DIM I AS INTEGER  
02. CAM.WRITEREG(17, 5) 'SPEED  
03. I = CAM.READREG(17)  
04. DEBUG.PRINT CSTR(I)
```

FULL LENGTH WORKING EXAMPLE

The BlazingCore Vision System is meant to be used together with the OLED display to make use of its high speed GRAM memory, thus eliminating the use of another external memory to store the frame (~140k). Data manipulation is done by reading the pixel data of a specific location from the GRAM into a variable stored in program memory. Of course, the OLED has to be initialised as well.

A full length code demonstrating frame captures from the camera and displaying them on the OLED is shown below.

Frame-grabbing and displaying to screen is currently done at 10 to 12 frames a second.

Code:

```
01. PUBLIC SUB MAIN()
02. 'Init OLED
03. OLED_INIT()
04.
05. 'Init Camera Module
06. CAM.WRITEREG(18, 128) 'Common Control A 0x12
07. DELAY(200)
08. CAM.WRITEREG(18, 64+32+16+8+4)
09. DELAY(200)
10. CAM.WRITEREG(17, 6) 'SPEED - DONT CHANGE 0x11
11. CAM.WRITEREG(19, 32+1) 'Common Control B 0x13
12. DO
13.     CAM.GRABFRAME()
14. LOOP
15.
16. END SUB
17.
18. '=====
19. PUBLIC SUB OLED_INIT()
20. OLED.INIT(0)
21. OLED.SETCOLOUR.FOREGROUND(31)
22. END SUB
23. '=====
24.
```

*For full hardware & software information, please refer to the *BlazingCore Vision System documentation*.

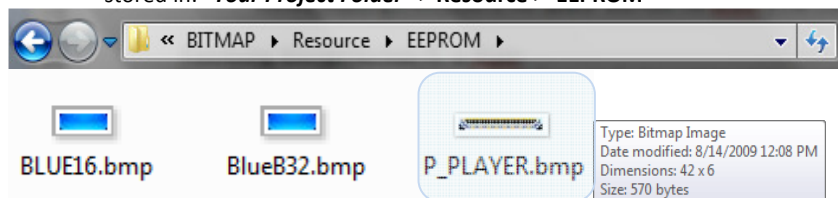
APPENDICES

TRANSFERRING BITMAP FILES FROM PC TO BCore

Store the Bitmap Files in your BCore Project's Resource Folder, and import it using the Resource Page in BCoreIDE. Files may be imported and downloaded into an external memory chip (DataFlash) or external memory card (MMC). Example code to draw bitmap files to the OLED from both external memory methods are available under its respective instruction set.

External DataFlash

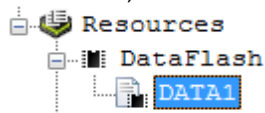
1. To store Bitmap files in the External DataFlash, make sure the Bitmap files you want loaded into the DataFlash is stored in: **"Your Project Folder" > Resource > EEPROM**



Example view using windows explorer.

Note: You may have to create these folders yourself if they do not exist.

2. Next, Select the data file (DATA1) under the DataFlash Resources of your project, and declare an array of type data.



```
DATA myData (arraySize) FromFile "FileName"
```

Code:

```
01. 'DATA1
02. DATA B_PLAYER(570) FROMFILE "P_PLAYER.bmp"
03. DATA BLUEB16(455) FROMFILE "Blue16.bmp"
```

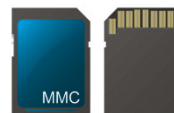
The array size should correspond to the size of your bitmap file in bytes (You can find out the size by placing your mouse over the particular bmp). (Refer to example as shown in figure above)

3. Compile and download the data file (DATA1) into the External DataFlash.
4. BMP files are now programmatically accessible through the use of the *variable name* (in this case, *B_Player*) as per the example code under the `OLED.Draw.BitmapDFlash` Command.

Note: The data file downloads directly to the external data flash chip, separate from downloading the actual program. Downloading your program does **NOT** download the data from the data file to the external data flash chip at the same time.

External Memory Card

1. To store Bitmap files in the External Memory Card (MMC/ SD Compatible), make sure you slot an MMC card into your card reader on your Laptop/PC and open it up using Windows® Explorer.
2. Drag and Drop your Bitmap files into the External Memory Card.
3. When you're done, eject the External Memory Card and slot it into the External Memory Card Socket onboard the OLED interface board.
4. The bitmap files in the External Memory Card are now programmatically accessible using the Bitmap File Name as per the example code under the `OLED.Draw.BitmapFromMemCard` Command.



*Please note that the file name length must be kept to 8 characters. Files with names longer than that may fail to open.

APPENDIX B

ASCII CHART

The following is a standard extended ascii chart supported by the BCore Font Library.

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99
100	101	102	103	104	105	106	107	108	109
110	111	112	113	114	115	116	117	118	119
120	121	122	123	124	125	126	127	128	129
130	131	132	133	134	135	136	137	138	139
140	141	142	143	144	145	146	147	148	149
150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169
170	171	172	173	174	175	176	177	178	179
180	181	182	183	184	185	186	187	188	189
190	191	192	193	194	195	196	197	198	199
200	201	202	203	204	205	206	207	208	209
210	211	212	213	214	215	216	217	218	219
220	221	222	223	224	225	226	227	228	229
230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249
250	251	252	253	254	255				

LATEST DOCUMENTATION

All of our documentations are constantly updated to provide accurate and/or new information that we feel would help you with developing with our products.

The latest documentation may be obtained from our website: <http://www.aiscube.com/main/downloads.html>

HOW YOU CAN HELP

You can help us to improve our documentations by emailing to us or posting a thread in our forum, reporting any mistakes/typos or errata that you might spot while reading our documentation.

Email: TechSupport@aiscube.com

Forum: <http://forum.aiscube.com/index.php>

DISCLAIMER

All information in this documentation is provided 'as-is' without any warranty of any kind.

The products produced by AIS Cube are meant for rapid prototyping and experimental usage; they are not intended nor designed for implementation in environments that constitute high risk activities.

AIS Cube shall assume no responsibility or liability for any indirect, specific, incidental or consequential damages arising out of the use of this documentation or product.

COPYRIGHT© 2009 - 2010 AIS CUBE. ALL RIGHTS RESERVED.

ALL PRODUCT AND CORPORATE NAMES APPEARING IN THIS DOCUMENTATION MAY OR MAY NOT BE REGISTERED TRADEMARKS OR COPYRIGHTS OF THEIR RESPECTIVE COMPANIES. AND ARE ONLY USED FOR IDENTIFICATION OR EXPLANATION FOR THE OWNER'S BENEFIT. WITH NO INTENT TO INFRINGE.

THE BLAZINGCORE(BCORE) AND BCORE IDE ARE TRADEMARKS OF AIS CUBE IN SINGAPORE AND/OR OTHER COUNTRIES. ALL IMAGES DEPICTING THE BLAZINGCORE OR ANY PART OF IT IS COPYRIGHTED.

ALL OTHER TRADEMARKS OR REGISTERED TRADEMARKS ARE THE PROPERTY OF THEIR RESPECTIVE OWNERS.